IDA*

$n + e$

Tsinghua University

2016 年 6 月 20 日

顾名思义

Iternative Deepening A*

↓

迭代加深 A* 算法

迭代深搜

- 给出一个限制 bound, 规定当搜索层数 $>$ bound 时直接剪枝
- 在最外层 for(bound), 如果无解就继续

迭代深搜

- 给出一个限制 bound, 规定当搜索层数 $>$ bound 时直接剪枝
- 在最外层 for(bound), 如果无解就继续
- 缺点: 重复计算

A 算法

- $f(n) = g(n) + h(n)$
- $g(n)$: 从起始状态到当前状态 n 的代价
- $h(n)$: 从当前状态 n 到目标状态的估计代价
- 估价函数 $h(n)$ 若选取不当，则可能找不到解，或找到的解也不是最优解.

A 算法

- $f(n) = g(n) + h(n)$
- $g(n)$: 从起始状态到当前状态 n 的代价
- $h(n)$: 从当前状态 n 到目标状态的估计代价
- 估价函数 $h(n)$ 若选取不当, 则可能找不到解, 或找到的解也不是最优解.
- 又 Wa 又 T 一时爽: CJK

h(n)

- 定义 $h^*(n)$ 为从当前状态 n 到目标状态的实际代价
- 必须满足 $h(n) \leq h^*(n)$, 否则嘿嘿嘿
- 估计总是过于乐观的

$h(n)$ 的相容

- 如果 h 函数对任意状态 s_1 和 s_2 还满足

$$h(s_1) \leq h(s_2) + c(s_1, s_2)$$

- $c(s_1, s_2)$ 是 s_1 转移到 s_2 的步数，则称 h 是相容的.
- h 相容能确保随着一步步往前走，f 递增，这样 A* 能更高效找到最优解.

$h(n)$ 怎么写?

- 不要太过分就好. 一般来说没问题的

$h(n)$ 怎么写?

- 不要太过分就好. 一般来说没问题的
- 脑洞要大

A* ?

- 比 IDA* 还麻烦……
- 听说还要 hash 判重和堆维护？每次要选取 $f(n)$ 最小的更新
- 时间和空间都不行

IDA*

- 在一般的问题中是这样使用 IDA* 算法的
- 当前局面的估价函数值 $h(n)+$ 当前的搜索深度 $g(n) >$ 预定义的最大搜索深度 bound 时，就停止继续往下搜索

IDA*

- 在一般的问题中是这样使用 IDA* 算法的
- 当前局面的估价函数值 $h(n)+$ 当前的搜索深度 $g(n) >$ 预定义的最大搜索深度 bound 时，就停止继续往下搜索
- 写成代码就是

```
if dep + h() > bound then return;
```

- 没了

Advantages

- 省时间省空间
- 试一下挺好写的

How to write IDA*?

```
int dfs (int dep) {
//dfs 返回大于 bound 的局面中最小的 f(n), 这样的话 bound 就
    int hv = state.h();
    if (hv == 0) return flag = 1, dep; //找到解
    if (dep + hv > bound) return dep + hv;
    int next_bound = 1000;
    for (int i = 0; i < 4; i++) {
        calc(new_state); int tmp = dfs(dep+1);
        if (flag) return tmp; //找到解
        if (tmp < next_bound) next_bound = tmp;
    }
    return next_bound;
}
for (bound = state.h(); !flag; bound = dfs(0));
```

运用

- 各种游戏 (1085, 1500, 2087) 求最少步数，普通搜索会爆炸

运用

- 各种游戏 (1085, 1500, 2087) 求最少步数，普通搜索会爆炸
- 1500 我用八数码 IDA* 是CX二进制压位 BFS 的 18.25 倍，Rank 1

运用

- 各种游戏 (1085, 1500, 2087) 求最少步数，普通搜索会爆炸
- 1500 我用八数码 IDA* 是CX二进制压位 BFS 的 18.25 倍, Rank 1
- 2087 我用双向广搜 + 二进制压位是CJK的 IDA* 的 2.25 倍, Rank 1

运用

- 各种游戏 (1085, 1500, 2087) 求最少步数，普通搜索会爆炸
- 1500 我用八数码 IDA* 是CX二进制压位 BFS 的 18.25 倍, Rank 1
- 2087 我用双向广搜 + 二进制压位是CJK的 IDA* 的 2.25 倍, Rank 1
- 不是说我写的就是快!!! 要学会选择合适的算法

1085 十五数码

- 尼玛……折腾了我整整 1 天. (几天前我还不会 DLX && IDA*)
- 套模板, 包括估价, 发现 T 了
- 发现 AC 的代码全都 Case 数据, 人神共愤!
- 于是改数据, 把答案范围改小, 时限调宽, 然而 233333

几个优化技巧

- 1. 位运算. $i \gg 2$, $i \,\&\, 3$
- *2. `#define abs(x) (x>=0?x:-(x))`
  一些经常调用的小函数拿去 *define* 掉, 因为声明函数栈空间花时间
  YHX 的程序原来 12.1s, 我加完后变成 8.4s
  平时玩玩可以, 考试时非常不推荐使用 (注意风险, 要加括号!!!)
- 3. 特判无解情况. 我放在题解里了
- 4. 二维数组改成一维, $a[4][4] \rightarrow a[16]$
- *5. 减少估价函数的计算量. 本题中的 $h(n)$ 满足加法运算
- 6. 调整 udlr 的顺序, 改变搜索序
- *7. 把起始局面和目标局面对掉. 我靠这个成功进 2s

几个优化技巧

- 1. 位运算. $i \gg 2$, $i \,\&\, 3$
- *2. `#define abs(x) (x>=0?x:-(x))`
  一些经常调用的小函数拿去 *define* 掉, 因为声明函数栈空间花时间
  YHX 的程序原来 12.1s, 我加完后变成 8.4s
  平时玩玩可以, 考试时非常不推荐使用 (注意风险, 要加括号!!!)
- 3. 特判无解情况. 我放在题解里了
- 4. 二维数组改成一维, $a[4][4] \rightarrow a[16]$
- *5. 减少估价函数的计算量. 本题中的 $h(n)$ 满足加法运算
- 6. 调整 udlr 的顺序, 改变搜索序
- *7. 把起始局面和目标局面对掉. 我靠这个成功进 2s
- 总结: 都是脑洞. 针对题目特点进行的优化最有效
- 大家还可以试试别的花样 $= =$